2001

An evaluation of a 2-way semijoin distributed query processing algorithm

Chen Chen
University of Windsor

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

NOTE TO USERS

This reproduction is the best copy available.

UMI®

An Evaluation of a 2-Way Semijoin Distributed Query Processing Algorithm

By

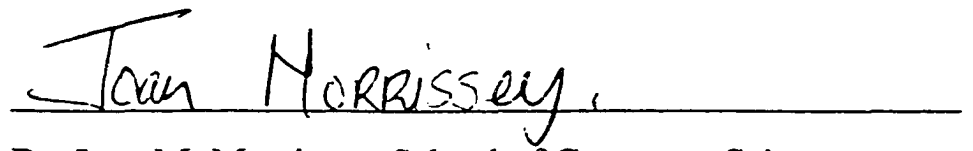
Chen Chen

A Thesis

**Submitted to the Faculty of Graduate Studies and Research
Through the School of Computer Science in Partial
Fulfillment of the Requirements for the
Degree of Master of Science at the
University of Windsor**

**Windsor, Ontario, Canada
2000**

APPROVED BY:

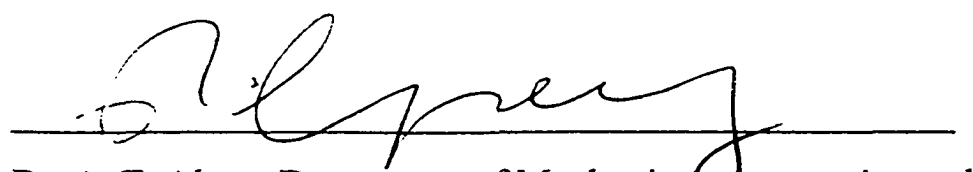


Dr. Joan M. Morrissey, School of Computer Science

Supervisor

Dr. Ezeife, School of Computer Science

Internal Reader



Dr. A. T. Alpas, Department of Mechanical, Automotive and Materials Engineering

External Reader

To my parents and my family...

ABSTRACT

The 2-way semijoin is proposed as an important extended version of the semijoin, which adds a backward reduction to maximize the reduction capability of traditional semijoin operations used as an effective operator for minimizing transmission cost in distributed query processing. In this thesis, we evaluate the 2-way semijoin algorithm objectively against a full reducer which is the algorithm that fully reduces all relations involved in a query by eliminating all non-participating tuples from relations. Instead of using filter-based algorithm, our algorithm is implemented so that it avoids hash collisions and also allows for composite semijoins. A series of experiments with various queries are carried out to study the above issues. It has been show that using our 2-way semijoin algorithm to reduce the query relations achieves significantly reduction effect. It performs well with respectable results on both the average percentage reduction of query relations and the percentage of queries that achieve full reduction in terms of total cost.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Dr. J. Morrissey for her academically inspiring guidance, supervision, suggestions and encouragement throughout my graduate study. I appreciate Dr. C. Ezeife whose advice, comments and patience on my graduate course study, the early stage of this thesis - 510 survey and this thesis work. I thank Dr. A.T. Alpas for his support, valuable comments and precious time.

I am also extremely indebted to my father ChunLin Chen, my mother XiuKun Liu, my husband Hong and my son Scott (Y. Zhang) for their support.

Finally, thanks to everyone who helped me.

CONTENTS

LIST OF FIGURES

Chapter 1 INTRODUCTION

In recent years, distributed database systems (DDBS) have drawn considerable attention from both the academic and industrial sectors due to their high functionality which integrates the advantages of two different technologies: the databases and the networks. As with any emerging technology, DDBS is more complex and the maintenance costs are much higher than that of the centralized database systems. The optimal processing of a distributed query over a number of different sites is a critical issue that affects the performance of the overall distributed database system. The problem of query optimization is much more difficult in a distributed environment due to the fact that data are resident at different sites.

1.1 Distributed Database Management System

A distributed database system refers to a collection of two or more databases which are dispersed geographically at a number of sites [ÖV99]. Each site locally consists of a single processor to maintain a local database. The computers at different sites are interconnected with one another by a communication network which allows for data sharing and distributed control. The integration of distributed databases increases system availability, reliability and promises better functionality and performance than offered by a conventional centralized database management system (DBMS). On the

other hand, some inherent problems derive from the system processing overhead because the transmitting of data over the network is necessary.

1.2 Distributed Query Processing

Retrieving data from different sites via a computer network is known as distributed query processing [AHY83]. Generally speaking, query processing implements the query optimization algorithms that translate queries into a series of data manipulation operations. Obviously, it often involves shipping a large number of relations and intermediate results between different sites. A query processing strategy determines a sequence of data movement and local processing in order to answer a query. Thus, processing queries with a minimized quantity of inter-site data transfers can significantly increase the performance of query optimization further, and improve performance of the overall system.

There may exist many strategies for a given query. The problem of distributed query processing is to find an efficient or optimal strategy to process queries among different sites. On the other hand, there is no single algorithm for all queries due to the diversity of DDBS environments such as, various networks, fragmented relations and so on. All of this makes query processing for distributed database systems significantly more difficult than for centralized database systems. Minimizing the cost of the query is a major issue in processing distributed query. This is the objective of query optimization in distributed database systems.

1.3 Query Processing Phases

Basically, a distributed query processing algorithm is defined to be a set of relational operations and network transmission steps. To process distributed queries, most algorithms follow the three major step paradigms.

1. *Initial local processing phase:* all local operations that require no intersite communication such as selections from relations, projections over the joining and target attributes and local join operations are performed at each site first.
2. *Reduction phase:* after the preprocessing by the first phase, a sequence of semijoins is derived from the remaining join operations and used as reducers to eliminated all unwanted tuples from the relations and therefore, the size of relations are reduced.
3. *Final processing phase:* all relations participating in the query may possibly be reduced by previous phases. They are sent to the final (or assembly) site where the final query processing is performed to produce the answer for a given query.

1.4 Problem Description and Thesis Objectives

As we know, communication is the dominant cost in distributed database, so minimizing the amount of data to be transferred is of prime importance. The problem is to find an optimal strategy which will reduce the relations efficiently. Several techniques are proposed in the literature to handle the potential processing bottlenecks that might occur before and during the final step of processing queries. The semijoin is one popular technique used to reduce the size of relations but with

high cost. To alleviate the problems of semijoins, recent research focuses on using filter based algorithms [Osb98][MM98][Lia99] to minimize the data transmission cost during distributed query processing. The major advantage of using filters instead of semijoins is that the filter is usually smaller than the join attribute projection. Thus, it speeds up local processing with lower network overhead compared with the semijoin. However, some filter based algorithms are limited to using perfect hash functions or are restricted to specific query types. Some relations in the query may not be reduced to the full extent possible due to the nature of hash collisions associated with the Bloom filter. Acutely, filter based semijoins can be viewed as a cheap implementation of the semijoin. This means that a relation being reduced by a Bloom filter may still contain tuples which would be eliminated by a semijoin. Unlike semijoins, filter based algorithms can not deal with composite semijoin at all, which can really reduce relations.

On the other hand, most semijoin algorithms proposed in this research area only do the semijoin in one direction only — forward reduction of relation refers to the traditional semijoin operation that leaves space for further improvement. The 2-way semijoin [KR87][RK91] is an important extended version of the semijoin and enhances the traditional semijoin with a backward reduction. However, no 2-way semijoin algorithm has been implemented or tested.

In this thesis, a 2-way semijoin algorithm is designed, implemented and evaluated. The primary goal of this thesis is to utilize a 2-way semijoin to achieve the maximum reduction capability of the semijoin operation. The experiments were conducted to test the performance of the algorithm against the full reducer specifically, the amount

of reduction in relations and the number of fully reduced queries under various test conditions. The objective of this thesis is to investigate what is the maximum reduction that 2-way semijoin can achieve. This evaluation work will lead to seek a right way for further improvement.

1.5 Outline of the Thesis

The rest of the thesis is organized as follows. Chapter 2 introduces some fundamental concepts and techniques of distributed query optimization, and summarizes some representative algorithms proposed in this research area. Chapter 3 describes our proposed algorithm with a running example. Chapter 4 explains the experimental system. In Chapter 5, the results of the evaluation of the algorithm are presented and discussed. Finally, the conclusions are given in Chapter 6.

Chapter 2 BACKGROUND INFORMATION

The mainstream of distributed query optimization still focuses on issues such as which operations will answer the query best and in what order they should be executed. However, to find an optimal or best strategy among all possible alternatives for a given query has been shown to be NP-hard [Hen80][KLK97]. Therefore, most query processing algorithms developed in this research area are based on various heuristics which give efficient, close to optimal solutions. The major approaches include joins [LMH85][LPP91], semijoins [AHY83][KR87][RK91] [PC90][MB97], a combination of joins and semijoins [CY93][CY92][CY94], dynamic methods [BRJ89][MBB95] and filters [Mul93][TC92][MM98][MO97][MBBK95][Lia99]. Current research focuses on using semijoins and Bloom filters.

This chapter will discuss some fundamental concepts and properties as well as the representative algorithms related to these techniques and mainly focus on the semijoin operations.

2.1 The Cost Models

Distributed query processing requires transmission of data among different sites in a computer network. For a given query, there may be many access plans that a database management system can follow to process it and generate the answer. All the plans are equivalent in terms of their final results but vary in their costs. To efficiently

perform query optimization in a distributed database system, the query optimizer uses the cost model to predict the cost of alternative execution plans for a global query and attempts to minimize this computing cost. There can be many factors that the cost model should take into account. In general, two factors are the local processing cost and the transmission cost. The local processing cost measures the time of CPU instructions and the time for disk I/O. The transmission cost refers to the time to transmit data from the site where they reside to the site where the computations are performed [ÖV99].

The two most used models are the *total time cost* model and the *response time cost* model. The total time cost is the sum of the local processing cost and the data transmission cost. However, in Wide Area Networks (WAN) such as the Internet, the transmission is the dominant time factor due to the shipment of data between the long distance sites therefore, the local processing cost is ignored. The total time cost model calculates the cost of data transmission only. The response time is the total execution time of the query from the beginning to the completion of the query in which any processing done in parallel is ignored.

Clearly, minimizing the total time cost can improve the utilization of the resources and increase the system throughput.

2.2 Join Based Approaches

The join is one of the fundamental operations that allows data stored at different relations to be combined together based on some common information in query processing. Given two relations R_1 and R_2. The common join attribute A exists in both relations. The join of R_1 and R_2 over A is denoted as $R_{1.A}\bowtie R_2$ and is performed by concatenating tuples of R_1 and R_2 where the values of attribute A are equal for both relations. The figure 2-1 gives an example of the join operation between relations R_1 and R_2.

The main properties of join operation include:

(1) It is an essential and critical operation.

(2) It functions as a selection and can be used as a reducer so that the size of join result is smaller than the relation size participated in join (see figure 2-1).

(3) It may also increase the result size (see figure 2-2). To transfer these results to another site is very expensive which impedes query optimization.

(4) It is time-consuming operation because in native implementation, each tuple in one relation must be compared to every tuple in the other relations.

R_1

A	B
a1	b5
a2	b7
a3	b9
a4	b3

R_2

A	C
a1	c8
a3	c4

R1⋈R2

A	B	C
a1	b5	c8
a3	b9	c4

Figure 2-1 The Join $R_1\bowtie R_2$ Over Attribute A

R_1

A	B
a1	b5
a1	b7

R_2

A	C
a1	c8
a1	c4

$R1 \bowtie R2$

A	B	C
a1	b5	c8
a1	b5	c4
a1	b7	c8
a1	b7	c4

Figure 2-2 Non-reduction Join

By using certain techniques, a join can be executed more efficiently in some computing environments. Various algorithms and execution models have been proposed in the literature to speed up the processing for the join operation, which include non-distributed and distributed joins [CY90][Mul93][Seg91]. [LPP91][CY90b] propose algorithms for determining a better join sequence to minimize data transmission costs. [CL89][AM91] optimized the two-way join in a broadcast local area network dealing with fragment relations. Joins can be executed in parallel in distributed systems [MPTW94].

The R* algorithm [LMH85] is a representative join-based approach developed for System R's optimizer. The objective of R* is to minimize the total time and it takes also into account local processing cost. It works in both local and wide area networks. In R*, a join between two relations is performed at a single site by using the "nested-loop method" or the "merge-scan method" which estimates the size of intermediate results in order to select the join order. The R* optimizer uses a dynamic program approach to generate new join sequence of n relations from join sequences of n-1 relations. Once the decision is made to join two relations, it follows by the decision for the more coming join relation candidates.

Another two methods, "shipped whole" and "fetch-match-matched" in R* are considered for transmitting relations. When the relation size is small, it is better to use "shipped whole". When the join relation is large, "fetch-match-matched" selects tuples as needed and only ships matching tuples.

The predicting and enumerating of these strategies is costly however. The exhaustive search may be too costly if the query is executed frequently. The complexity of R* is exponential.

2.3 Semijoin Based Approaches

2.3.1 Definition and Properties

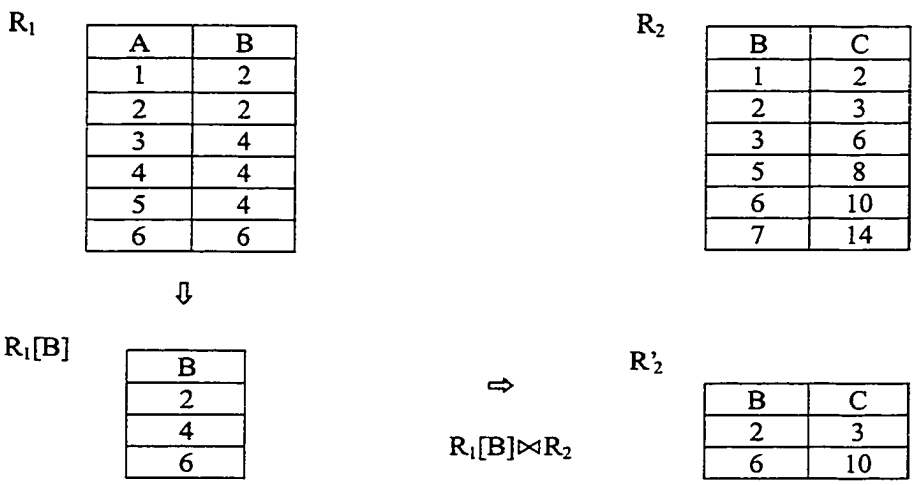
The semijoin is first introduced by [BC81][BGW81] as an effective operator to alleviate the cost of expensive joins in data transmission and has been used as a size reducer to minimize the cost of processing distributed queries. The objective of the semijoin is simple. By eliminating those tuples which will not be part of the join result, the semijoin reduces the size of the relation first then, the join is performed on the reduced relations which will minimize the data transmission cost.

Given two relations R_1 and R_2, stored at different sites, a semijoin from relation R_1 to relation R_2 over the common joining attribute B is denoted $R_1 \ltimes_B R_2$ and executed as follows.

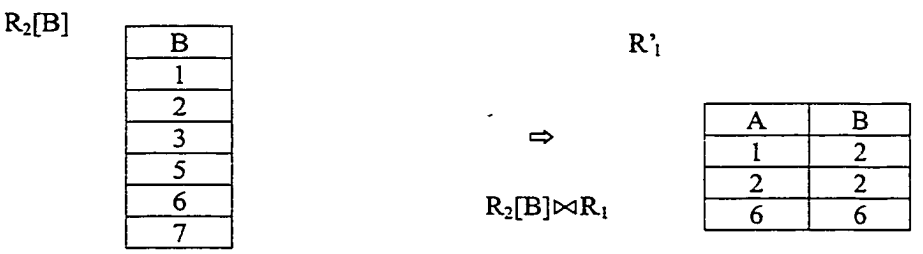
1. Projects R_1 over attribute B to get projection $R_1[B]$
2. Ship the projection $R_1[B]$ to the site of relation R_2

3. Perform $R_1[B] \bowtie R_2$

The example of semijoins is shown as figure 2-3.



a. semijoin $R_1 \ltimes {}_BR_2$



b. semijoin $R_2 \ltimes {}_BR_1$

Figure 2-3 Semijoin Example

In the above, the first two steps refer to the reducing phase which needs extra processing cost due to the construction of semijoin projections. The step 3 is the joining phase which gives reduction benefit. The cost of semijoin is mainly the cost of the reducing phase. Therefore, if the projection size is smaller and much less than the relation size, the semijoin results in a sufficient reducer. A sequence of semijoins, called a semijoin program can be used further to reduce the size of the operand relation. A carefully chosen sequence of semijoins can significantly reduce the sizes of the relations before they are shipped to the query site. One strategy used in query processing is to choose the relation with the least in_degree and process it first. The use of the semijoin operations has led to the development of full-reduction methods for processing a distributed query [BC81][Ozs80][BGW81].

The properties of the semijoin operation include:

(1) It is cheaper than the join. A semijoin can be computed with less inter-site data transfer by eliminating unjoinable tuples.

(2) It will never increase the size of result relation, with the worst case being no reduction.

(3) It is not commutative $(R_1 \rtimes R_2) \neq (R_1 \ltimes R_2)$. The example is shown in figure 2-3.

(4) There is local processing overhead because an additional transferring of the join attribute projections is needed to perform a semijoin.

(5) There are propagation effects which means the reduction effect of a semijoin can be propagated to reduce the costs of other semijoins when a sequential semijoin operation is performed.

(6) There is no information loss because of collisions.

(7) It allows composite semijoins with more than one common join attribute which gives more reduction power.

(8) It does not work for all kinds of queries. Cyclic queries are hard to process.

(9) It is not good for using in local area networks where the communication cost is not the dominant factor.

2.3.2 The Research based on the Semijoins

Most distributed query optimization algorithms rely on some variants of the semijoin technique [BC81][BGW81][WCL93][MB95][RK91]. The major concern in semijoin processing is to reduce the inter-site data movement with the least total transmission cost. Some issues focus on seeking the optimal semijoin programs for these classes of queries which can be fully reduced by semijoins such as, tree queries. Some well-known query optimization algorithms based on semijoin strategies are summarized in this section.

2.3.3 SDD-1 Algorithm:

SDD-1 [BGW81] is the improved implementation of the first distributed query processing algorithm [EW77] and is based on semijoins. The objective of the SDD-1 approach is to minimize total communication time by using semijoins as reducers. It uses database profit, the statistics associated with a relation as input to generate a set of beneficial semijoin (BS) execution strategies as output.

The SDD-1 algorithm proceeds in four phases. First, all local processing such as appropriate selections and projections are performed at each site. Statistics are updated to generate a set of beneficial semijoins in the initial phase. In the next phase, a greedy algorithm is used to select the most beneficial semijoin and modifying the statistics. The following phase is to select the assembly site. The costs of transferring all the required data to each candidate site are evaluated in this phase. The last phase, postoptimization, will remove unnecessary semijoins from the execution strategy which is stored at the assembly site.

The drawback of SDD-1 is that it ignores higher-cost semijoins which may result in benefit for other semijoins when the strategies are generated at each step.

2.3.4 AHY Algorithm:

Apers, Henver and Yao present a collection of semijoin based algorithms [AHY83], which are based on their previous algorithms [HY79] Algorithm SERIAL and Algorithm PARALLEL.

Algorithm GENERAL is capable of optimizing general queries and consists of three versions; two designed to minimize the total time and one to minimize the response time. For all three versions, a general query is decomposed into simple queries by isolating each join attribute. AHY algorithm follows three steps:

1. Apply all local processing to reduce the amount of data to be transferred.
2. Generate candidate schedules either to minimize response time or total time for a simple query using Algorithm PARALLEL or Algorithm SERIAL respectively.

3. Integrate the decomposed queries from step 2 into a near optimal execution strategy

Algorithm PARALLEL begins with computing an initial feasible solution to the query. By trying to join small relations to large relations, Algorithm PARALEL computes a minimum response time schedule for each join attribute individually to search for cost beneficial data transmissions.

Algorithm SERIAL is employed to reduce the total cost. It attempts to keep the cost of shipping relations as low as possible. For each relation, it selects the best semijoin schedule with minimum transmission cost. The relations are shipped in ascending order of size to the next site.

Generally speaking, Algorithm GENERAL is a great improvement over Algorithm SERIAL and Algorithm PARALLEL. In theory, this algorithm can be applied to any general distributed query environment if all assumptions are satisfied. Although AHY allows parallel transmission of semijoin projections, the parallel generation of semijoin projections and parallel generation of semijoins are not considered. The Response Time Version does not consider query delays and synchronization issues in parallel systems, while the Total Time Version might result in redundant data transmission. The collective version is based on a greedy strategy.

2.3.5 One-Shot Algorithm:

Previous semijoin algorithms perform semijoins sequentially, in which the reduction effect of a semijoin can be propagated to reduce the cost of the following semijoins. However, this may increase the processing overhead when the sequential execution of

semijoin is in some order such as, $R_i \rtimes R_j$, $R_i \rtimes R_k$. In this case, relation R_i has to be scanned twice to semijoin with R_i, R_k respectively. which is inefficient. The sequential semijoins also lead to a loss of parallelism. Processing semijoins simultaneously at all sites can remedy the inefficiency problem of sequential processing semijoins strategies.

The one-shot semijoin approach [WCS92][WLC93] makes use of all different forms of parallelism for minimizing the query response time such as parallel generating of all semijoin projects, the parallel transmission of all the semijoin projections and the parallel execution of all the semijoins.

Najjar and Slimani proposed a new approach [NS99] which extended the one-shot semijoin strategy to minimize data transmission cost in distributed query processing. The method is based on a combination of join and parallel semijoin operations. The extension of the one-shot semijoin first focused on finding an optimal set of profitable semijoins. Then, it applied a sequence of joins as reducers to reduce the cost of intersite data transmission and to move data in parallel. The authors assume that the amount of data to be transferred is large and the speed of transmission is set to 1 therefore, the transmission setup cost was ignored.

2.3.6 Domain-Specific Semijoin:

There are some restrictions if using semijoins in a fragmented database. The semijoin can only be used in relation to relation, or relation to fragment manner but not fragment to fragment which causes elimination of contributive tuples. The Domain-Specific Semijoin [CL90] was proposed as an improved semijoin operation. It uses

the semantic information associated with the joining attribute to deal with the problems related with semijoins between the fragments and to ensure that no data is lost in the final result. The authors integrate the domain-specific semijoin with an existing semijoin strategy and derive cost and benefit estimation formulas for calculating the cost effectiveness of a domain-specific semijoin. It is more flexible and more cost-effective than traditional semijoin in most cases.

The strategy proposed goes as follows:

1. Use a cost function to calculate the estimated cost and benefit of the candidate semijoin.
2. If profitable, put it into the current query-processing strategy otherwise, drop it.
3. Update the profile with the suggested formula if a domain-specific semijoin is included.

Although this strategy attempts to take advantage of the domain information to avoid unnecessary processing, domain information is not always available and not easy to be specified.

2.3.7 2-Way Semijoin:

The semijoin is an effective operator to reduce the cost of data transmission. Much work has been done to optimize query processing in distributed databases based on this strategy. However, the traditional semijoin only does a forward reduction of the relations which leaves space for further improvement. To maximize the reduction capability of a semijoin, the 2-way semijoin is introduced [KR87][RK91]. As an important extension version of the semijoin, the 2-way semijoin enhances the

traditional semijoin with a backward reduction capability which results in the reduction of both relations involved. Therefore, it has more reduction power than the traditional semijoin.

For $R_i \bowtie R_j$ with join attribute *A*, the 2-way semijoin is usually implemented as follows:

(1) Send projection $R_i[A]$ from site *i* to site *j*.

(2) Execute $R_i[A] \bowtie R_j$ to reduce R_j. During this, partition $R_i[A]$ into $R_i[A]_m$ and $R_i[A]_{nm}$. $R_i[A]_m$ contains the set of values in $R_i[A]$ which match a value of $R_j[A]$, while $R_i[A]_{nm}$ is $R_i[A] - R_i[A]_m$.

(3) Send either $R_i[A]_m$ or $R_i[A]_{nm}$, which ever is smallest back to the site of R_i.

(4) Reduce R_i. If $R_i[A]_m$ was shipped in (3), execute $R_i[A]_m \bowtie R_i$. Otherwise, use $R_i[A]_{nm}$ to eliminate tuples from R_i whose attribute A are matching one of those in $R_i[A]_{nm}$.

In above procedures, the step (2) is referred to forward reduction and step (4) as backward reduction.

The traditional semijoin can be viewed as one way semijoin. It reduces a relation R_i to contain only those tuples that match with the tuples from another relation R_j. The 2-way semijoin simultaneously reduces both relations R_i and R_j so that they each contain only tuples that match the other relation. It was pointed out that the 2-way semijoin strategy produces the most reduction when all forward reduction is performed before any backward reduction [RK91][LR95]. It is particularly useful when both relations R_i and R_j need to be transmitted to the third site. Using a 2-way

semijoin with tuple connectors and a pipeline cache planner to process an N-way join algorithm is also introduced in [RK91].

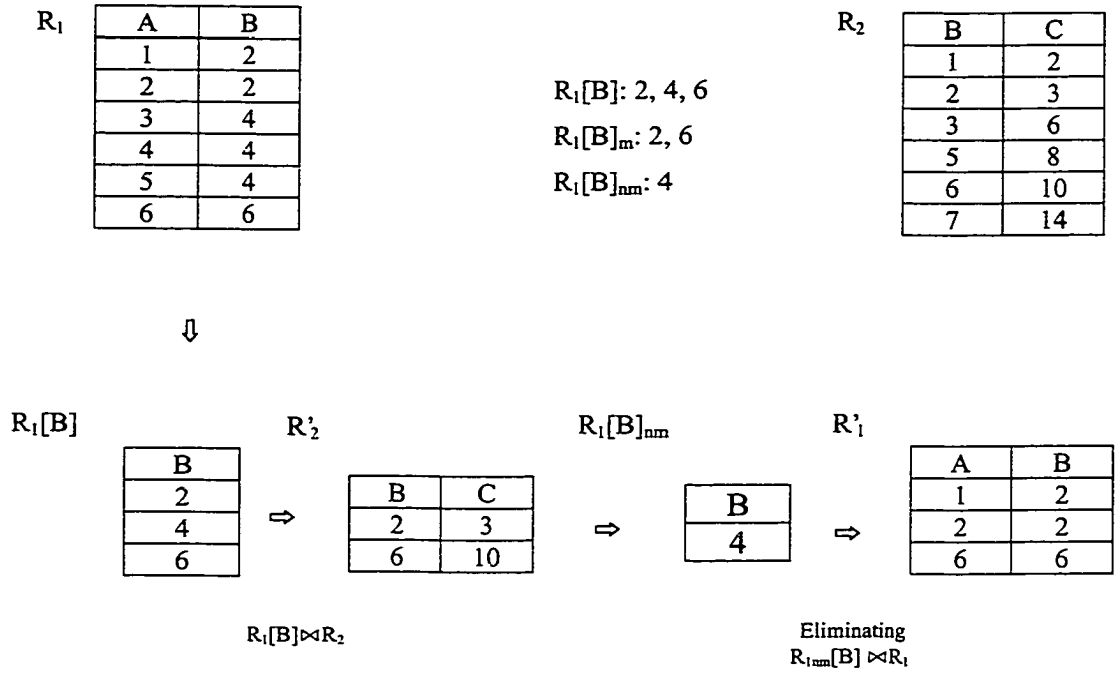


Figure 2-4 2-Way Semijoin Example

2.3.8 Intelligent Search Method

Yoo and Lafortune presented an intelligent search approach [YL89] which employs the A* heuristic research method [BF81] to determine a sequence of semijoin reducers for distributed query processing. Their algorithm avoids the drawbacks of conventional ways that either use a hill-climbing algorithm to find an efficient but not

optimal solution or use dynamic programming to find an optimal solution that requires expanding all the database states in the search space.

The key issue of the A* algorithm is to derive a heuristic function which can intelligently guide the search of a sequence of semijoins and avoid unnecessary efforts on exploring unpromising sequences of reducers. To guarantee that the estimate function can find the optimal sequence of semijoins, A* algorithm proves the satisfaction of admissibility and consistency conditions. Admissibility ensures that an optimal path from any state to the final state is found and consistency ensures that the path chosen from the initial state to any other state is optimal.

By immediately deleting states that will never be on an optimal path, their method enhances the efficiency of the A* [BF81] algorithm. The concept of an A* search can be applied to cyclic queries. But no details are given in this paper.

2.3.9 Algorithm W

Morrissey *et al.* [MBB95][Bea95][MB97] propose two semijoin based algorithms, called Algorithm W and Algorithm DW which attempt to minimize data transmission costs during distributed query processing. Algorithm W is a static heuristic which uses the concepts of profit, marginal profit and gain to determine effective schedules of semijoins for the construction of reducers. In this phase a reducer is generated from cost effective semijoins for each common join attribute. The schedule is also executed by constructing the reducers in parallel. Algorithm DW is a purely dynamic version of Algorithm W which uses up to data information eliminating the need for schedule monitoring. For each join attribute participating in the query, a reducer is

created and applied one at a time. However, the greedy approach used by Algorithm DW considers the execution of a semijoin with no form of "look ahead" except for the calculation of marginal profit. The decision to begin with the cheapest semijoin may not lead to the construction of the best reducer. Consequently, the performance of dynamic Algorithm DW is not better than the static semijoin Algorithm W in most case. The performances of these algorithms are superior to AHY algorithm.

2.3.10 Some Drawbacks of Semijoin

Through the discussion above, the semijoin is better at reducing relation sizes and is much cheaper than the join operation in most cases. However, some class of queries can not be fully reduced by semijoins alone. The semijoin approach is better if the semijoin acts as a sufficient reducer, that is, a few tuples of relation R participate in the join. Performing semijoin operations might entail more local processing costs.

2.4 Filter Based Approaches

By adopting the tuple bit vector idea rather than the commonly used semijoin projections, filter based algorithms as alternatively semijoin approaches have been employed as an alternative to the semijoin. A filter, also called bit vector filter is a hash-based array used to encode information about the values contained in an attribute. The filter functions as a compact representation of the values of a join attribute. Transferring the bit array instead of the join attribute values to the join site saves communication time. Various filters can be constructed in similar way but are

used differently. Recent research focuses on the use of Bloom filter [MO97][MM98][Mul93][MBBK95].

2.4.1 Bloom Filter

A Bloom filter [Blo70][Bab79] is simply an array of bits generated by using a hash function on a join attribute. Usually, bloom filters are used as a reducer to minimize some cost in distributed query processing. It can achieve the same result as a semijoin does but with lower data transmission costs. A filter can be constructed as follows:

1. Construct an array and initial all bits to zero.
2. For each value of the join-attribute use a hash function to produce an address.
3. For each address produced, set the corresponding bit to one.

The following example shows how a filter works as a reducer.

Suppose two relations R_i and R_j have common join attribute b. $R_i \bowtie R_j$ are processed as follows:

1. Constructing a filter for attribute b of R_i
   a. for each tuple in R_i, hash on the value for attribute b to produce an address
   b. for each address produced, set to1
2. Ship filter to the site of R_j.
1. For each tuple in R_j
   a. Hash on the value of attribute b to produce an address
   b. If the address produced in the filter is set to one then accepting the tuple as part of answer, zero is rejected.

It has been shown that Bloom filters can filter out the tuples which cannot be the part of the result relation (figure. 2-5). Compared with semijoin projection, the size of a filter is generally smaller than the size of join attribute. Obviously, it is cheaper to transfer a filter over the network than a semijoin projection. The use of bloom filters can significantly reduce data transmission costs and local processing is also cheaper. However, the hash collisions can result in a loss of join information. Some work [WCS92] has shown that it is not always optimal to reduce relation R fully with Bloom filters. The Representative Algorithms are [MO97][MM98][Lia99].

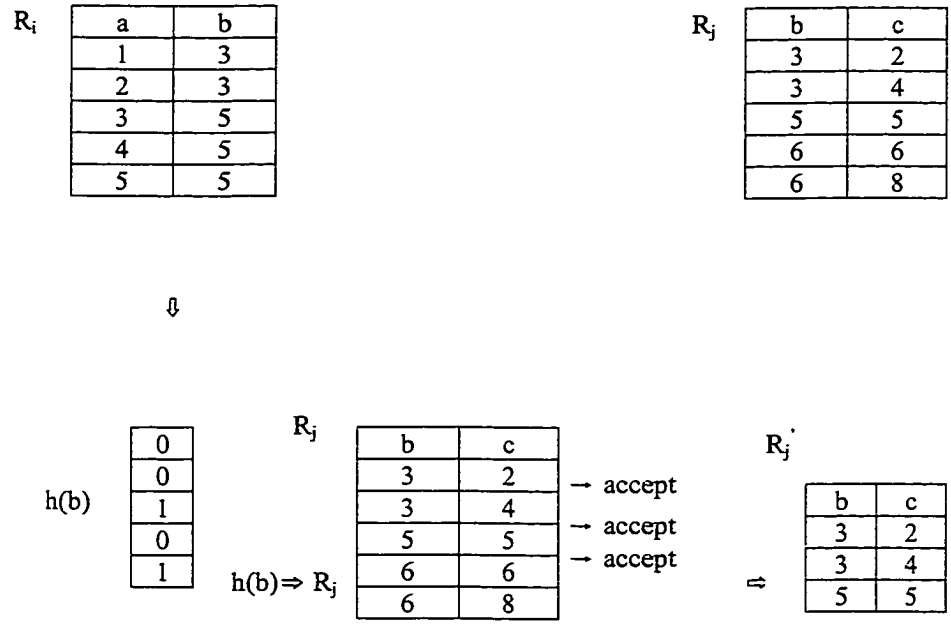


Figure 2-5 The Reduction of R_j by the Reduction Filter for Attribute b

2.4.2 Algorithm X

Morrissey and Ma proposed a heuristic called Algorithm X [MM98] to minimize the response time in query processing. This algorithm is completely based on Bloom filters and concerned with general queries. The objective of Algorithm X is to minimize the response time as well as local processing costs.

Algorithm X ships all relevant filters to every involved relation concurrently in order to achieve parallel transmission and application of filters. Each relation is processed twice. First, the algorithm hashes for all join attributes to produce addresses in their corresponding filters. Second applies these filters to reduce the relation concurrently. The experimental evaluation shows that this algorithm results in the significantly reducing both the response time and local processing cost. The limitation of this algorithm is the assumption of the perfect hash function.

Morrissey and Osborn [MO97] propose another algorithm using reduction filters to minimize the cost of data transmissions for general queries. The main phases in algorithm are *construction of reduction filter* and *processing of queue.* The algorithm is evaluated against a full reducer. It accomplishes the same reduction effects as semijoins with less cost. The use of a perfect hash function is also assumption in this heuristic which is unrealistic. Further work has been done in [Lia99]. In there, two sets of filters are employed instead of one for each joining attribute to reduce the collisions. Therefore, the performance of the algorithm improves.

Although the filter based algorithms have lower transmission costs than semijoins, it suffers collisions that happened when two or more attribute values hash to the same address.

Chapter 3 The ALGORITHM

We evaluate a 2-way semijoin algorithm based on the 2-way semijoin operator [KR87]. The algorithm uses attribute projections as reducers to remove unjoinable tuples during query processing and attempts to maximize the reduction effect of the semijoin by adding a backward reduction capability, which will result in reducing the size of relations for both relations involved in the semijoin.

The algorithm can process general queries consisting of an arbitrary number of relations and join attributes. Each relation is usually only processed once to minimize data transfers. However, if an attribute projection changes during use then we do a backward semijoin to reduce the appreprate relation. We apply the algorithm for general queries to investigate our hypothesis: without hash collision, it will fully reduce more relations. The implementation details will be given in this chapter.

3.1 Hypothesis

The semijoin has been introduced to reduce the cost of processing distributed queries due to its good reduction property. The drawback is higher communication cost. The use of filters can achieve the same result as a semijoin with lower cost. However, it suffers collision due to the use of hash functions. In order to process a query effectively, it is necessary to enhance the semijoin reduction with less transmission cost. We notice that numerous semijoin algorithms proposed in this research area do

only forward reduction of the relations which leaves space for further improvement. The 2-way semijoin as an important extended version of semijoin adds a backward reduction to reduce relations in both directions. It maximizes the reduction capability of the semijoin operation. Besides, using semijoin projections will avoid data loss caused by hashing collisions. Several lemmas and corollaries have proved that 2-way semijoin is much better than the traditional semijoin in reducing relation size theoretically, but the detailed implementation issues have not been addressed in the original paper [KR87]. We will evaluate a 2-way semijoin algorithm and would like to investigate our hypothesis. Without hash collisions, the performance of our 2-way semijoin algorithm will be better since the 2-way semijoin produces more reductions in the relation. The possibility of processing composite semijoins can lead to more reduction in the relations than filter based algorithms.

We will present a detailed explanation of a method we developed to evaluate the 2-way semijoin heuristic in terms of a full reducer.

3.2 Details of Implementation of the Algorithm

3.2.1 Query Graph

To better process queries, a join query graph is used to represent the queries. In the query graph, Vertices represent a set of relations. Edges are the joining attributes. The relations referenced in the query are assumed to be located at different sites. The in degree is the number of common joining attributes with the other relations. For

example, relation R_1 has in-degree number as 3 since it has three common joining attributes C with R_4, joining attribute A with R_2 and B with R_3.

Figure 3-1 gives an example of query graph.

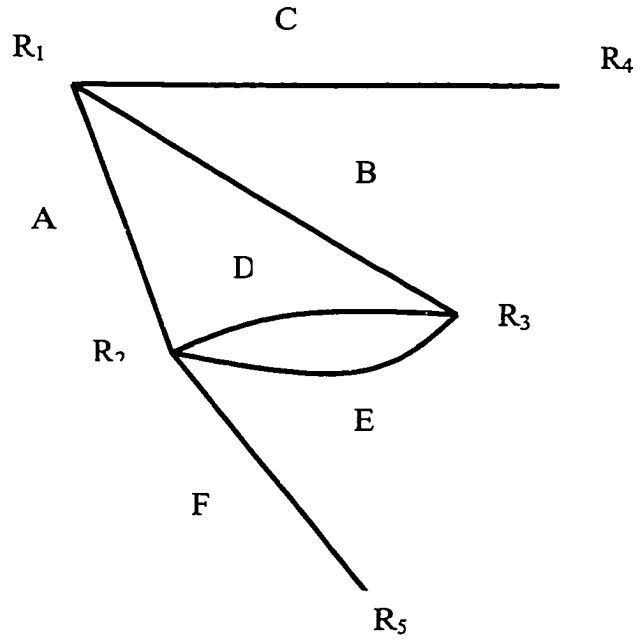


Figure 3-1 The example of query graph

Each query described by a graph is stored in an adjacency list in order to be processed by our algorithm. Based on this list, we can determine the order for processing relations using 2-way semijoin algorithm. We follow the semijoin rule to do the backward semijoin whenever the attribute projections are changed. We describe our algorithm as follows.

3.2.2 The Algorithm

Repeat the steps below until all relations processed.

1. Determine schedules for construction of projections by select relation with lowest in-degree to process. Mark this relation as R_i.
2. Scan adjacency list of R_i in order to construct all its projections. Denote the relation which has common joining attribute with R_i in this adjacency list as R_j. Check Projection_list (P_list) to see if any attribute exists in the list. If it is not empty, the attribute projection has been constructed already.
   a. If projection exists, using existed attribute projection semijoin with relation R_i to reduce it and project out all the attributes from reduced relation R_i to constructed all attribute projections concurrently. Put projected attributes in the P_list. If any value of attribute projection has been changed, make a mark on it.
   b. Otherwise, construct all projections for R_i then put these attributes in the P_list
3. Apply semijoin rule only do the 2-way (backward) semijoin if projections have changed
   a. If attribute has been changed. Using updated attribute projected backward semijoin with R_j. reduce R_j and produce new attribute projections.
   b. Update P_list.
4. Reduce the in-degree of adjacent relations by 1
5. Mark relation as processed.
6. Repeat all steps until all relations in the query have been proceed.

Note that when we reduce a relation using attribute projection, we consider that if relation R_i has more than one common joining attributes with R_j, which means when multiple joining attributes appear in one relation, we will do composite semijoin with the relation supposed to be reduced. It results concurrently to reduce a relation more efficient than applying the same attributes independently. Therefore, it can increase reducing ability. We use all available existing projections to reduce and process the relation simultaneously. This is a great improvement. When all join relations are minimized, the final joins are optimal referring to the amount of data transmission.

3.3 An Example of using the Algorithm

To demonstrate the performance of our algorithm, we give an example as follows. Five example relations R_1, R_2, R_3, R_4, and R_5 are shown in the figure 3-2 are represented in our test database. The query to be processed requires executing join operations among all the relations on the common joining attributes, $R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4 \bowtie R_5$. The entire database is represented in terms of the graph-based data structures, which consist of an Adjacency matrix, Adjacency lists and Projection-list.

Figure 3-2 is example of test database.

R_1

A	B	C
1	2	4
2	2	5
3	3	6

R_2

A	D	E	F
1	2	4	3
2	2	5	5
3	3	6	7
4	4	7	9

R_3

B	D	E
2	2	4
3	3	6
4	7	9

R_4

C	G
4	2
5	3
7	4

R_5

F	H
3	4
3	7
7	7

Figure 3-2 Example Database

An Adjacency matrix is used to show the attributes of the relations in the query

	A	B	C	D	E	F	G	H
R_1	1	1	1					
R_2	1			1	1	1		
R_3		1		1	1			
R_4			1				1	
R_5						1		1

Figure 3-3 Adjacency matrix

Based on Adjacency matrix, we construct an Adjacency-list (A-list) for each relation, The head node includes the relation name and the number of in-degree of the relation, which is used to decide the order for construction of projections. Each joining attribute with relevant relation name is stored in the other nodes respectively. When all projections of joining attributes have been completely constructed for a relation, the number of indegree is changed to special mark –1.

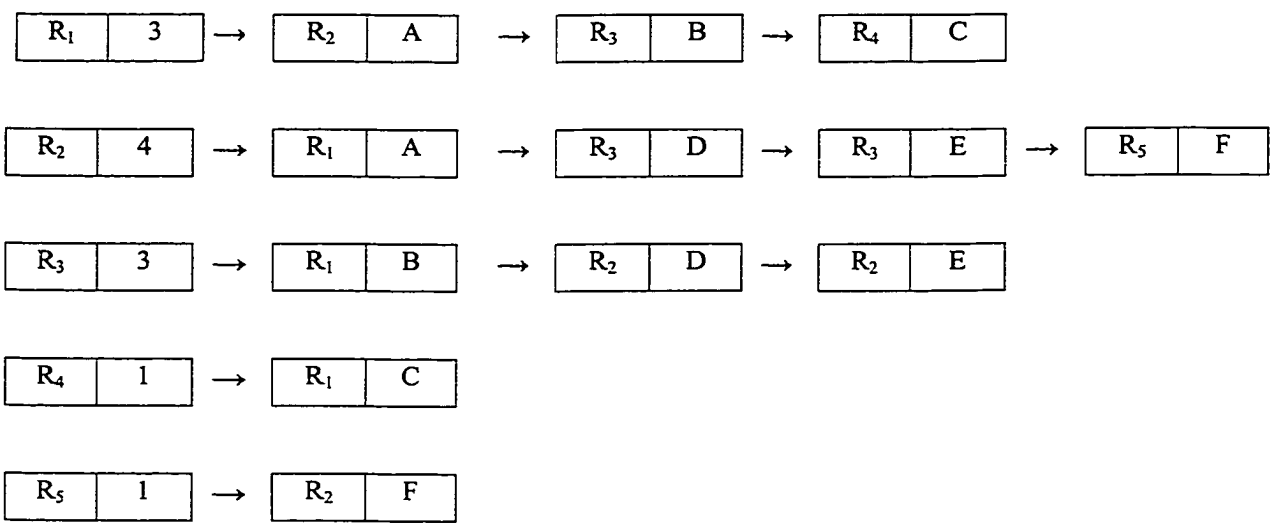


Figure 3-3 The adjacency lists

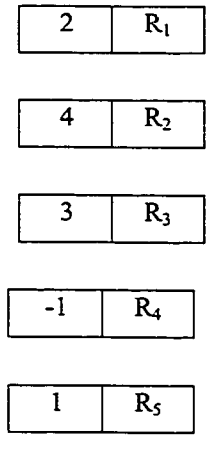
Projection list (P-list) shows the projections which have been produced. They are ready to be used.

The algorithm performs the query as following steps:

Step 1: Determine schedules for construction of projections.

- We scan the adjacency lists and select the relation with lowest in-degree to construct projections for each join attribute. The first relation to be processed in this case will be either R_4 or R_5 since they both have the lowest in-degree 1. We select R_4. The attribute projections need to be constructed are $R_4[C]$.
- Checking P-list, it is empty. We do projection over joining attribute C and get $R_4[C]$: 4, 5, 7. Put the C in the P-list to mark that projection [C] has been constructed.
- Reduce the in-degree of R_4 adjacent list. The indegree of R4 changed is to –1 to indicate that all attributes has been constructed completely. This relation is processed already. The indegree of relation R_1 also needs to be decreased by 1 since it has a joining attribute C with R_4 in common.

The figure 3-4 shows the updated head nodes of the A-lists and other data structures after processing Step 1.



P-list : C

$R_4[C]$: 4, 5, 7

R_4	C	G
	6	2
	7	3
	7	4

Join relation: R_1

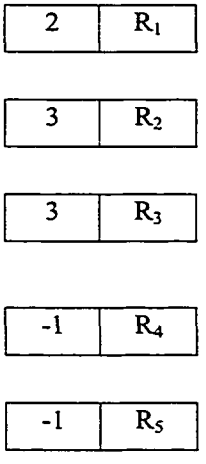
Figure 3-4 The example of Step 1

Step 2: We continue the process, scan the A-list, select relation R_5 with lowest in-degree 2 to produce attribute projection [F] for R_5.

- Checking P-list which shows no projection for attribute F available, we project R_5 over F and get projection R_5[F]: 3, 7.
- Put F in P-list since its projection has been produced. Update A-list head notes as it processed already.
- Reduce indegree of join relation R_2 by 1.

The changes made to the data structures after processing step 2 are illustrated below.

The head nodes of A-list:



P-list : C, F

[C]: 4, 5, 7

[F]: 3, 7

R_5

F	H
4	4
4	7
7	7

Join relation: R_2

Figure 3-5 The example of Step 2

Step 3: Scan adjacency lists and select relation with lowest in-degree R_1 to construct its attribute projections [A],[B] and [C].

- Check P-list. Since projection R_4[C] has already been produced we simply use it to semijoin with R_1 and do projections over attributes A and B from reduced R_1 simultaneously. The value of new Projection R_1 [C] has been change to 4, 5.
- Based on semijoin rule, we execute semijoin [C] $\ltimes$ R_4, to reduce relation R_4.
- Update P_list, and indegree numbers of the join relations involved.

The updated data structures and reduce relations shown as Figure 3-6

The head nodes of A-list:

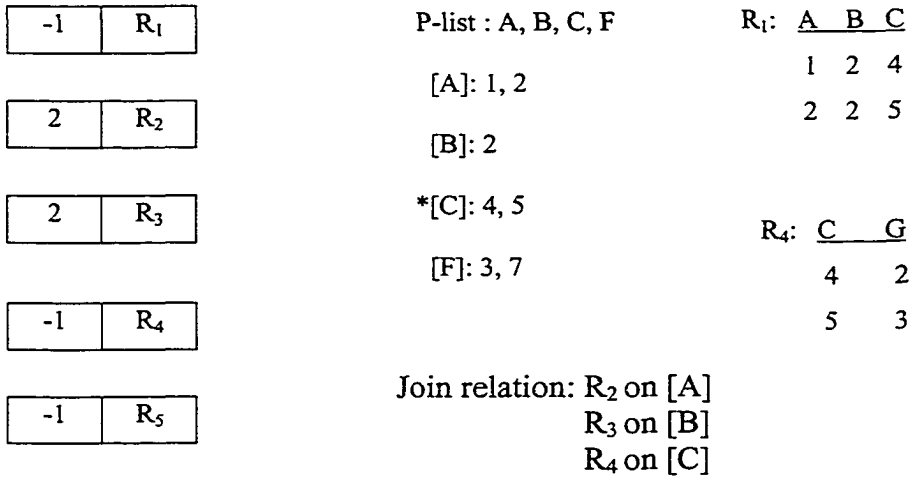


Figure 3-6 The example of Step

Step 4: Scan a-list, we select R_2.

- Scan P_list, select R_2. It has attributes [A], [D], [E] and [F].
- Check P_list, since [A] and [F] have existed, We use them to reduce R_2, execute $[A] \rtimes R_2$ and $[F] \rtimes R_2$ first.
- Project new attribute projections from reduced relation R_2. We check the updated value of projections [A] and [F] have been changed.
- Follow semijoin rule. Do backward semijoins $A \rtimes R_1$ and $F \rtimes R_5$ to reduce the relation R_1 and R_5.
- Projection $R_1[C]$ has been changed again, do 2-way (backward) semijoin to reduce the relation R_4
- updated the data structures and the indegree of relations

The figure 3-7 shows the final result after processing relation R_2

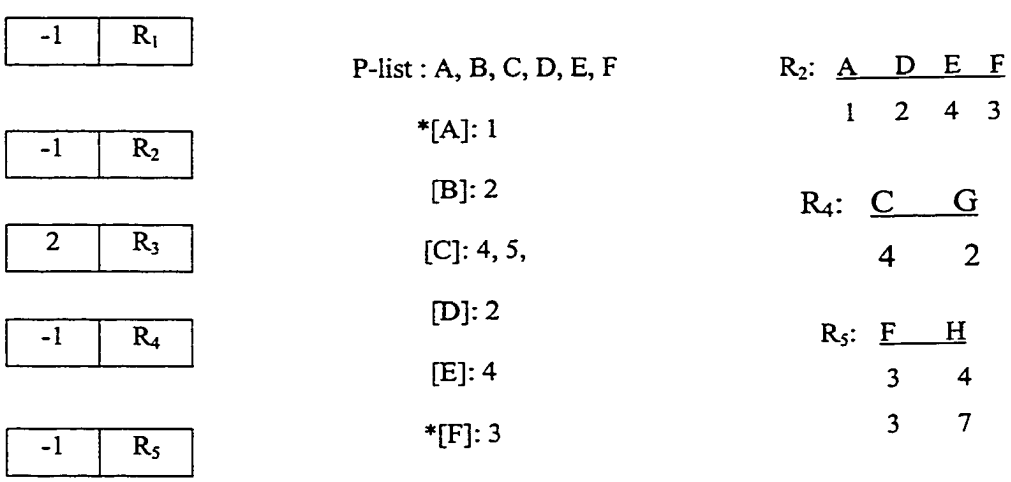


Figure 3-7 The example of Step 4

Step 5: Scan A-list, we select R_3 to process.

- Check the P_list for attributes [B], [D] and [E]. Since all projections are available. We do not need to rebuild them, just use them to reduce relation R_3.
- Get updated attribute projections.
- Since all projections are not changed. No backward semijoin is required. Update indegree number.

The results after processing Relation R_3 is shown as figure 3-8

:

P-list : A, B, C, F

[A]: 1,

[B]: 2

[C]: 4

[D]: 2

[E]: 4

[F]: 3

R_3:

B	D	E
2	2	4

Figure 3-8 The example of Step 5

Since all relations in A-list has been processed. The process stops.

The set of reduced relations is shown as figure 3-9. All the relations are fully reduced the algorithm.

R_1:

A	B	C
1	2	4

R_2:

A	D	E	F
1	2	4	3

R_3:

B	D	E
2	2	4

R_4:

C	G
4	2

R_5:

F	H
3	4
3	7

Figure 3-9 Fully reduced the relations by the algorithm

Chapter 4 EVALUATION

Evaluating the relative performance of the heuristic needs the both theoretical analysis and empirical study. In this chapter, we prove the methodology for our experimental work. We describe a series of experiments that we ran on our implementation of a 2-way semijoin algorithm. The experiments we ran are designed to evaluate the validity of assumptions made and the techniques used.

4.1 Methodology

The framework for evaluating our algorithm is based on the objectives of semijoin processing:

- To compare the performance of our algorithm against the effects of full reducer. This experiment is designed to test what number of relations are fully reduced under various test conditions. It will illustrate how close the algorithm achieves fully or optimal reduction in relations.
- To measure the performance enhancement of the algorithm over IFS in terms of total cost. This experiment is used to test the cost of the algorithm under different test conditions when processing distributed queries. It will illustrate the total transmission cost that the algorithm requires for executing queries.

We assume the relational data distributed among all the sites is neither fragmented nor replicated. We will only consider select-project-join (SPJ) class of queries. It will not be a limitation since the possibility exists to translate most queries into SPJ type.

Our algorithm can process general queries, which consist of an arbitrary number of relations and joining attributes.

4.2 Evaluation Framework

A collection of software is used to conduct the experiments. It is consists of the evaluation framework such as, query-relation generator; the full reducer; Initial Feasible Solution (IFS) measurement, our 2-way semijoin algorithm and some corresponding analysis programs. For efficiency reasons, we implement the algorithm with C++. All the application programs are tested in Microsoft Visual C++ 6.0.

4.2.1 The Query Generator

To evaluate our algorithm, the first experimental work we need is to establish the test database. In all cases, the query_relation generator we are using is the one developed by the Database Research Group at School of Computer Science at University of Windsor [BWT95]. It is based on the Wisconsin Benchmark relations [BDT83] with modifications to be more useful in selecting attribute domain values to testing general queries. The C program "creat_query.c" and "relbuilder.c" are developed to produce the actual relations. Specifically, by adopting the statistical representation of queries, the "creat_query.c" generates the statistical tables. The inputs of the program consist of the number of relations and the number of common join attributes involved in the query to be generated. The outputs include the file of database statistics "dbstates, the file of attributes domain size "domains" and several relation files "Rel_i" which corresponds to each relation in the query. These "Rel_i" files in turn, as input files are

subsequently used by program "relbuilder.c" to construct the output relation files "r_i" referenced as relations in the statistical table. Each relation file "r_i" contains the required number of tuples, the number of join attributes and the join attribute key as well as their values. All these relation files consists test database for evaluating purpose. The major advantage of using statistical technique is that we only need to construct the relations which are participating in the query rather than the entire database. The figure 4-1 is an example of query statistic table. In where, two numbers 5 3 in the first line represent the number of relations and the number of common joining attributers included in the query respectively. All the statistics for each relation specified in the query are stored from the second line. The first column $S(R_i)$ is the size for each relation R_i. Followed by the cardinality S(dij) and the selectivity ρ(dij)for each attribute(dij) in each relation.

In relation file "Rel_i", the first number (510) is the size of the relation. The second one (2) is the number of joining attributes in this relation. Then a set of numbers including attribute key (0), attribute size (125) and attribute domain size (150) for each attribute followed one by one. In the example, another set of numbers is for attribute (1), its size (201) and its domain size (230).

dbstats

5 3

$S(R_i)$	$S(d_{i1})$	$\rho(d_{i1})$	$S(d_{i2})$	$\rho(d_{i2})$	$S(d_{i3})$	$\rho(d_{i3})$
200	0	0.000000	164	0.745455	135	0.750000
300	122	0.762500	218	0.908333	235	0.940000
360	122	0.762500	188	0.783333	0	0.000000
290	0	0.000000	188	0.783333	0	0.000000
500	0	0.000000	228	0.950000	245	0.980000

Rel_i

510 2 0 125 150 1 201 230

Fig 4-1 Example of "dbstats" and "Rel_i" files

For simplicity, only integer numbers are considering by query_relation generator. It generates queries containing three to six relations and two to four joining attributes.

4.2.2 Full Reducer

The comparison between two algorithms can show the improvement of one algorithm over another. It is the common approach used in previously proposed algorithms [MBB95][MO97][PC90] for evaluation. However, it is hard to show how close the algorithm comes to achieve full or optimal reduction in relations. In order to evaluate the algorithm objectively, we use a full reducer and compare the performance of our algorithm against a full reducer. The main reason is based on the important issue in query optimization on semijoin reduction [CH80]:

For a given relation, there exist several potential semijoin programs. There is one optimal semijoin program, called the full reducer, which for each relation R reduces R more than the others. What a full reducer really does is eliminate all non-participating tuples from the relations first, then ships these tuples to achieve full reduction. Therefore, by using this experimental approach to evaluate our algorithm, we can determine how close it achieves full reduction under various conditions. It is better than just against the any other algorithms.

The full reducer program is designed as follows.

1. Join all relations required by the query to get the result.

   Using a nested loop join, we join all relations together to obtain all candidates of attributes and relations appearing in the result relation.

2. Obtain the reduced relations by projecting the attributes of each relation from the joining result.

   Any unsatisfied tuple in each projected relation would be eliminated. Therefore, It is reduced fully.

The figure 4-2 shows the example of full reducer. The relations we using are shown as figure 3-2.

First, join all relations involved in the query. Figure 4-2 shows the result of joining five relations together.

$R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4 \bowtie R_5$

A	B	C	D	E	F	G	H
1	2	4	2	4	3	2	4
1	2	4	2	4	3	2	7

Second, we project all attributes out for each relation from the result of joining the five relations. By doing this, all non-participating tuples are eliminated from each relation. Therefore, the relation achieves fully reduced. Each reduced relation referred as full reducer for that relation.

R_1 full reducer:

A	B	C
1	2	4

R_2 full reducer:

A	D	E	F
1	2	4	3

R_3 full reducer:

A	B	C
2	2	4

R_4 full reducer:

C	G
4	2

R_5 full reducer R_5:

F	H
3	4
3	7

Figure. 4-2 Example of Full Reducer

4.2.3 Initial Feasible Solution (IFS) Measurements

The first join strategy used in distributed database systems is called the initial feasible solution (IFS). It ships all relations directly to the query site and performs join there. The cost of IFS is simply summing all the cost of transferring all relations to the joining site.

In order to determine how significant improvement of the algorithm is, we compare the cost of our algorithm with IFS. This experimental approach is designed based on the another issue in semijoin processing: besides to measure the capability of full reduction we also would like to minimize the total transmission cost [Wang90].

We assume that if transmission cost can be expressed in time units which means transferring one value of data to joining site costs one unit, the total cost will be the

sum of the units of all relation data to be shipped to the joining site. Therefore, the cost of IFS is adding the sizes of all relations participating query without any reduction.

*Relation Size = Number of Tuples * Number of Join Attributes*

For our algorithm, we reduce the relations before shipping to the joining site. Clearly, the cost of our algorithm is the cost of each attribute projection shipped plus the size of reduced relations added. We should take the both factors into account to calculate our cost. To measure the cost of our algorithm over IFS, we based on the formula as follows.

Total Cost of the Algorithm over IFS =

(Total Cost of IFS – Total Cost of the Algorithm) / Total Cost of IFS

4.3 The Experiments

To evaluate the performance of our algorithm precisely, two major experiments we employ as mentioned before are:

1. Comparing our algorithm against full reducer
2. Measuring the cost of the algorithm over IFS

In order to carry on the comparisons, we set up different test conditions for experiments and wish to see the affections on the performance of our algorithm.

Based on the test database, following characteristics of a set of test queries and relations are examined.

(1) the number of relations involved in a given query

(2) the number of possible attributes involved in a given query

(3) the selectivity of the attributes

(4) the domain sizes of attributes

(5) the cardinality of the relations

4.3.1 Query Types

The queries generated by query_relation generator made up of many combinations referred to different query types. For example, query type 6_4 represents that there are six relations with totally four joining attributes in that query. In total, twelve different query types will be represented in the experiments. The range is from the query type 3_2 to the query type 6_4.

4.3.2 Selectivity

Selectivity is defined as the number of distinct values of that attribute divided by the number of the attribute domain size. Using attribute selectivity one can predict the reduction effect of the semijoins. The values of an attribute are randomly chosen from the domain of that attribute in our experiments. For example, assuming the domain size for attribute A is 1000, hence the possible values will be in 0-999. If attribute A is to have a selectivity value of 50%, then 500 different values will be randomly selected from domain poll. These are the actual values of attribute A. Joining attribute

has high selectivity if the ratio is low and the low selectivity if the ratio is high which means the good selectivity is only a few matching attributes.

4.3.3 Percentage of Reduction

An another experiment we using to evaluate the algorithm are calculating the average percentage of reduction achieved by our algorithm and the percentage of queries that achieve full reduction. The calculation is based on the formula as follows.

*Reduction (%) = [(total size - reduced size) / (total size - full size)] * 100*

In the formula above:

- *total size* is define as the size of all relations related to the query before reduction
- *reduced size* represents the size of the relations which have been reduced by the algorithm.
- *full size* are the size of the relations which have been reduced by fully reduced by the reducer

Here is example:

Total size =59

Reduced size = 38

Full size = 34

Reduction (%) = [(59 - 38) / (59 - 34)]*100 = 84%

The full reduction (%) represents the number of queries which are full reduced by the algorithm. The number of queries is out of 100 queries.

4.3.4 Individual Runs

The performance of the algorithm is evaluated with over 3600 queries that vary in 12 different query types and 3 different selectivity. For each input combination, we run 100 queries. To handle such a huge number of diverse queries, it is necessary to split the queries into runs.

To classify the various comparisons, we divide three testing groups based on different selectivity. The ranges of selectivity we chose are

(1) 0.02— 0.4

(2) 0.4 —0.7

(3) 0.7—0.95

They will cover attribute selectivity from low to high and indicate the effect on the performance of the algorithm.

We subdivided test queries based on the query type again. Therefore, 12 different query types will be tested in the same range of selectivity. 100 different queries are included in each query type as one run. Totally, we have 36 runs in the relations, joining attributes and selectivity. This will help us to conclude that with the same selectivity, what the performance is on different query types.

In additional, the attribute domain size is 150 — 250, the number of tuples in each relation is 200 — 600.

The results from these experiments are presented in the next chapter.

Chapter 5 RESULTS

Based on our experimental efforts, we present the implementation result for evaluating 2-way semijoin algorithm in this chapter. The results can be classified into two categories in order to show the performance of the algorithm on two import issues — the reduction ability and the total cost. The observations are from the comparisons between the performance of our algorithm against the full reducer as well as the performance of our algorithm over IFS in terms of total time cost. We give a discussion on analysis some result issues.

5.1 Results of the Reductions

The main purpose of our algorithm is to reduce the number of tuples involved as much as possible during processing distributed queries (referred to maximum reduction ability of the relations). To evaluate the performance of our algorithm on this point, we design experiments by comparing our algorithm against a full reducer. We conduct experiments varying in the selectivity, the query types in order to demonstrate how these factors will impact the performance of our algorithm. We would like to see if the number of relations, the number of attributes and the appearance of these attributes really matter for optimal queries.

The figure 5-1 shows the experimental results.

In this table, there are 12 different query type from 3—2 to 6—4. Each query type have been test with different selectivity ranged 0.02 — 0.4, 0.4 — 0.7and 0.7 — 0.95. Reduction (%) represented average percentage reduction produced in each type of queries by the algorithm. Fully Reduced (%) is the percentage of queries that fully reduced by the algorithm on each query type. The results are from each run respectively. The bottom line of the table shows the result of average reduction for all query types that are calculated from each column.

	Selectivity 0.02—0.4		Selectivity 0.4—0.7		Selectivity 0.7—0.95	
Type	**Reduction**	**Fully Reduced**	**Reduction**	**Fully Reduced**	**Reduction**	**Fully Reduced**
3—2	98.29	65	98.18	60	96.16	62
3—3	96.67	82	91.30	78	86.09	67
3—4	99.45	91	99.04	88	98.59	77
4—2	98.49	85	98.98	80	98.17	59
4—3	96.47	79	95.85	70	89.41	56
4—4	97.89	90	96.10	84	91.50	72
5—2	98.94	89	98.99	90	98.48	73
5—3	98.70	92	98.71	87	99.75	77
5—4	98.26	87	97.33	78	94.40	72
6—2	99.53	92	99.39	89	98.39	76
6—3	98.41	89	97.54	79	98.86	84
6—4	99.16	86	96.73	76	93.59	66
Average of the Column	98.60	86	97.35	80	95.28	70.08

Figure 5-1 Comparison with Full Reducer

We analyze the experimental results based on figure 5-1. For all cases, average reduction are calculated as:

Average reduction = (98.6 + 97.35 + 95.28) / 3 = 97.08%

Average fully reduction = (86 + 80 + 70.08) / 3 = 78.69%

We summarize as follows:

- On average, approximately 97.08% of all unneeded tuples are eliminated by the algorithm from the relations before shipping them to the query site. This result indicates the algorithm achieves substantial reductions in relation sizes. And also the results show that, on average, the algorithm fully reduces the relations in 78.69% of all relations which achieved an higher percentage of full reduced queries with all cases.
- When joining attribute selectivity varies from different ranges on the queries that include all the query types, the results show that there is no substantial difference on the average reduction of the algorithm. In the best case (selectivity 0.02 — 0.4), average reduction achieves 98.60% while in the worse case, (selectivity 0.7 — 0.95) the average reduction is 95.28%. The slight difference is kept within 3.32%.
- For average fully reduction, the best average percentage of fully reduction achieves 86% (selectivity 0.02 — 0.4). The worse case of the average fully reduction is 70.08% (selectivity 0.7 — 0.95). The difference between the best case and the worse case is 15.92%. This is really factor affects the full reduction. We should consider.

- Comparing the average reduction on each query type with varying selectivity, the actual result show the best average reduction is 99.53% with query type 6 – 2 (selectivity 0.02 — 0.4). The worse case of average reduction is 86.09% with query type 3 – 3 (selectivity 0.7 — 0.95) which can still eliminate a substantial unwanted number of tuples involved in the query. On the other hand, the best percentage of fully reduced relation is 92% with query type 5 – 2 (selectivity 0.02 — 0.4) and worse case 60% with query type 3 – 2 (selectivity 0.4 — 0.7). This 26.09% difference will affect the reducing query relation fully.

	Reduction 0.02 — 0.4	**Reduction 0.4 — 0.7**	**Reduction 0.7 — 0.95**	**Average**
3 — 2	98.29	98.18	96.16	97.54
3 — 3	97.67	91.30	86.09	91.69
3 — 4	99.45	99.04	98.59	99.03
4 — 2	98.89	98.98	98.17	98.68
4 — 3	97.47	95.86	89.41	94.25
4 — 4	98.48	96.10	91.50	95.36
5 — 2	98.94	98.99	98.48	98.80
5 — 3	98.70	98.71	99.75	99.05
5 — 4	98.26	97.33	94.40	96.66
6 — 2	99.53	99.39	98.39	99.10
6 — 3	98.41	97.54	98.86	98.27
6 — 4	96.16	96.73	93.59	95.49

Figure5-2 Average Reduction on Each Query Type

Further, we analysis the results of Reduction and Fully reduced vary on each query type.

- From figure 5-2, comparing average percentage reduction on each query type with varying selectivity, the type 3 – 3 has the lowest reduction on average 91.69%. The type 6 – 2 has the highest amount of reduction, 99.01%. This indicates the algorithm can reduce the relations for whatever the query type with expected result. The performance of the algorithm is efficient.

	Fully Reduced 0.02 — 0.4	**Fully Reduced 0.4 — 0.7**	**Fully Reduced 0.7 — 0.95**	**Average**
3 — 2	65	60	62	62
3 — 3	82	78	67	76
3 — 4	91	88	77	85
4 — 2	85	80	59	75
4 — 3	79	70	56	68
4 — 4	90	84	72	82
5 — 2	89	90	73	84
5 — 3	92	87	77	85
5 — 4	87	78	72	79
6 — 2	92	89	76	86
6 — 3	89	79	84	84
6 — 4	86	76	66	76

Figure5-3 Average Fully Reduced on Each Query Type

- From figure 5-3, we notice except query type 3 – 2, all the others have not much difference on the result of fully reduced. However, the query type 3 – 2 has the lowest fully reduction 62%. Comparing with the query type 6 – 2 (the highest fully reduction 86%), there is 24% difference. This is substantial. If the percentage of queries that achieve full reduction increase on this type, the overall full reduction will get increased.

	Selectivity 0.02—0.4	Selectivity 0.4—0.7	Selectivity 0.7—0.9
Type	Our algorithm over IFS	Our algorithm over IFS	Our algorithm over IFS
3—2	0.73	0.58	0.18
3—3	0.80	0.67	0.56
3—4	0.79	0.74	0.70
4—2	0.88	0.75	0.47
4—3	0.90	0.67	0.45
4—4	0.91	0.78	0.67
5—2	0.85	0.58	0.22
5—3	0.82	0.60	0.26
5—4	0.93	0.80	0.69
6—2	0.87	0.61	0.44
6—3	0.92	0.77	0.29
6—4	0.94	0.81	0.72
Average of the Column	0.86	0.70	0.47

Figure 5-4. Comparison with IFS

5.2 Results of the Costs

A comparison of the algorithm to the IFS is expected to answer the question: how well the algorithm performs in terms of total cost. We based on the formula (described as 4.2.3) to measure the average cost of our algorithm over IFS. The experiments are conducted under various test conditions. The actually results are shown as figure 5-4.

In this figure, the queries are tested from query type 3 – 2 to the type 6 – 4 with the selectivity varying (selectivity 0.02 — 0.4), (selectivity 0.4 — 0.7) and (selectivity 0.7 — 0.95). Each data in this table is the results of 100 queries. We analyze the these experimental results and summary as follows:

- In all cases, average cost of our algorithm over IFS

  $= (0.86 + 0.70 + 0.47) / 3 = 0.676$

  $\approx$ 68%

- The query type 3 – 2 with selectivity (0.7 — 0.95), has the lowest average total cost over IFS 18%. It has few numbers of the relation and few numbers of the attribute involved in the query. The query type 6 – 4 with selectivity (0.02 — 0.4) has the highest average total cost over IFS 94%.
- In the group of the selectivity (0.02 — 0.4) and the group of selectivity (0.7 — 0.95), for all the query type with the same selectivity range, it seems that the more relations and the attributes included in the query the better improvement of the algorithm over IFS cost. For example, the query type 6 – 4 is significantly over the cost of query type 3 – 2, as the number of relations and the number of

attributes go to high. The algorithm performances better with less cost. However, this is not the factors to affect the performance in the group of selectivity (0.4 — 0.7).

- For each query type however, as the selectivity changes from low to high, the average total cost over IFS drops from 0.73 – 0.18 (73% – 18%) in worse case with query type 3 – 2 from selectivity (0.02 — 0.4) to the selectivity (0.7 — 0.95). The least average cost over ISF is 0.18 (18%) with query type 3 – 2 in the group of the 0.7 — 0.95. However, with 0.68% of the average total cost over IFS on average in all cases is significant.
- For all queries, the group selectivity 0.02 — 0.4 has the most average cost over IFS which achieves 86%, on average 70% with the selectivity 0.7 — 0.95 and the lowest over IFS 47% in selectivity 0.7 — 0.95.

Chapter 6 CONCLUSIONS

In this thesis, we present the issue of evaluating a 2-way semijoin strategy for optimizing distributed queries. The most significant advantage of the 2-way semijoin approach over the traditional semijoin method is that it increases the reduction ability by adding backward reduction affecting both relations involved in the queries. Furthermore, it reduces the transmission costs. Our primary goal of evaluating the 2-way semijoin algorithm is to utilize the maximum reduction ability of semijoin to reduce the relation sizes with less transmission costs.

In order to experimentally measure how close our algorithm comes to achieving full reduction of relations, we evaluate our algorithm by a series of experiments under various test conditions in order to determine how close it comes to achieving full reduction of relations. Based on the experimental results, we conclude:

- The performance of the algorithm on average produces 97.08% average reduction in all cases which results in the relation sizes being significantly reduced by eliminating unneeded tuples before shipping them to the query site. 78.69% of all queries are fully reduced.
- On average the reduction of the algorithm in total cost over IFS is 68% for all queries which is a significant improvement over the IFS.
- The results show that varying the selectivity, the number of relations and the number of the attributers in the queries do not really matter with respect to performance of the algorithm but they do affect the average total cost over ISF.

With more relations and more attributes involved in the queries, the algorithm is more efficient than IFS in terms of totals. This result provide the fact that a high selectivity factor requires a large number of tuple comparisons, produces a larger result relation and requires more cost than the low selectivity does.

The performance of our algorithm has been evaluated on two important issues: the reduction ability of semijoin and the its total cost objectively by comparing to a full reducer and ISF. Our results indicate that our 2-way semijoin can reduce the number of tuples involved in the query substantially and efficiently reduce the size of relations.

REFERENCES

[AHY83] P. M. G. Apers, A. R. Hevner, and B. Yao *"Optimization algorithms for distributed queries"*, IEEE Transactions on Software Engineering, 9(1), 57-68, 1983.

[AM91] J.K. Ahn and S.C. Moon, "Optimizing joins between two fragmented relations on a broadcast local network", Info. Sys.16(2),185-198, 1991.

[Bab79] E.Babb, "*Implementing a relational database by means of specialized hardware*", ACM Trans. on Database Systems, Vol.4(1), 1-29, 1979.

[BC81] P.Bernstein, D.Chiu "*Using semi-join to solve relational queries*", Association for Computing Machinery Journal, Vol.28, 25-40, Jan 1981.

[BDT83] D. Bitton, D.J. DeWitt, and C. Turbyfill, "Benchmarking database systems a systematic approach", In Proc. Of the 9th VLDB Conf., 8-19, Nov.2, 1983.

[Bea95] W.T. Bealor, "Semijoin strategies for total cost minimization in distributed query processing". Master Thesis, University of Windsor, 1995.

[BF81] A.Barr and E.A.Feigenbaum, Ed., Handbook of Artificial Interlligence, Vol. I. Los Altos, CA: Willam Kaufmann Inc., 1981.

[BGW81] P.Bernstein, N.Goodman, E.Wong, C.Reeve, and J.Rothnie, "*Query processing in a system for distributed databases (SDD-1)*", ACM Trans. on Database Systems, Vol.6(4), 602-625, 1981.

[Blo70] B.H. Bloom, *"Space/time trade-offs in hash coding with allowable error"*, Communications of the ACM, 13(7), 422-426, 1970.

[CL89] J.S.J Chen and V.O.K. Li, "*Optimizing joins in fragmented database systems on a broadcast local network*", IEEE Trans. Software Eng. 15(1), 1989.

[CL90] J.S.J. Chen and V.O.K Li, "Domain-specific semijoin: a new operation for distributed query processing", Information Sciences, V.52, 1990.

[CY90] M.S. Chen and P.S. Yu, *" Using combination of join and semijoin operations for distributed query processing"*, In Proceedings of the 10th International Conference on Distributed Computing Systems, 328-335, 1990.

[CY92] M.S. Chen and P.S. Yu, *"Interleaving a join and semijoin operations for distributed query processing"*, IEEE Transactions on Parallel and Distributed Systems, 611-620, 1992.

[CY93] M.S. Chen and P.S. Yu, *"Combining join and semijoin operations for distributed query processing"*, IEEE Transactions on Knowledge and Data Engineering, 534-542, 1993.

[CY94] M.S. Chen and P.S. Yu, *"A graph theoretical approach to determine a join reducer sequence in distributed query processing"*, IEEE Trans. on Knowledge Data Eng. 6(1), 152-165, 1994.

[CY90b] M.S. Chen and P.S. Yu, *" Using join operations as reducers in distributed query processing"*, In Proceedings of the 2th International Symposium on Database in Parallel and Distributed Systems, 116-123, 1990.

[EW77] E. Wong, "Retrieving distributed data from SDD-1: A system for distributed databases", In Proc. 2nd Berkeley Workshop Distributed Data Management and Compute. Networks, 217-235, May 1977.

[Hen80] A.R.Henver, *"The optimization of query processing in distributed database systems"*, PhD thesis, Purdue University, 1980.

[HY79] A.R. Hevner and S.B. Yao, "Query processing in distributed database systems", IEEE Trans. Software Eng., vol.SE-5, no.5, 177-187, May 1979.

[KLK97] H.Kim, S.Lee and H.Kim, *"Distributed query optimization using two-step pruning"*, Information and software technology Vol.39, 149-169, 1997.

[KR86] H.Kang, N.Roussopoulos, "*On Propagation of Reduction Effects of a Semijoin in Distributed Query Processing*", Technique Reports: TR 86-53, 1986.

[KR87] H.Kang and N.Roussopoulos, *"Using 2-way semijoins in distributed query processing"*, In Proceedings of The 3rd International Conference on Data Engineering, 664-651,1987.

[Lia99] Y. Liang, "Reduction of collisions in bloom filters during distributed query optimization". Master thesis, University of Windsor, 1999.

[LMH85] G.M.Lohman, C.Mohan, L.M.Haas, D.Daniels, B.G.Lindsay, P.G.Selinger, and P.F.Wilms. *"Query processing in R*"*. Query processing in database system. Springer, New York, 1985.

[LPP91] P.Legato, G.Paletta and L.Palopoli, "*Optimization of join strategies in distributed database"*, Infor. Systems Vo.16(4), 363-374, 1991.

[LR95] Z. Li and K.A. Ross, "*PERF join: an alternative to two-way semijoin and bloomjoin*", In Proceedings of CIKM'95, 137-144, 1995.

[MB95] J.M. Morrissey and S. Bandyopadhyay, *Computer communication technology and its effect on distributed query optimization strategies",* In Proceedings of the Canadian Conference on Electrical and Computer Engineering, 598-600, 1995.

[MB97] J.M. Morrissey and W.T. Bealor, "Minimizing data transfers in distributed query optimization: A comparative study and evaluation", Computer Journal, vol.39, no. 8, 1997.

[MBB95] J.M. Morrissey, S. Bandyopadhyay and W.T. Bealor, "*A heuristic minimizing total cost in distributed query processing*", In Proceedings of the 7th International Conference on Computing and Information — ICCI'95, July 5-8, 736-758, 1995.

[MBBK95] J.M. Morrissey, S. Bandyopadhyay, W.T. Bealor and S. Kamat, "Dynamic strategies and bloom filters for minimizing data trandfers in distributed query optimization. 1995.

[MM98] J.M. Morrissey and X. Ma, "Investigating response time minimization in distributed query optimization", In Proceedings of ICCI's 98, 1998.

[MO97] J.M Morrissey and W.K. Osborn, "*Experiments with the use of reduction filters in distributed query optimization*", In Proceedings of 9th IASTED International Conference on Parallel and Distributed Computing and Systems, 1997.

[MPTW94] C.Mohan, H.Ptrahesh, W.G.Tang, Y.Wang, "*Parallelism in relational database management systems*", IBM Systems Journal 33(2), 349-371, 1994.

[Mul93] James K. Mullin, *"Estimating the size of a relational join"*, Information Systems 18 (3), 189-196, 1993.

[NS99] F. Najjar and Y. Slimani, " *Extension of the One-Shot Semijoin Strategy to minimize Data Transmission Cost in Distributed Query Processing*". Information Sciences 114(1-4), 1-21, 1999.

[Osb98] Wendy. K. Osborn, "The use of reduction filters in distributed query optimization". Master thesis, University of Windsor, 1998.

[ÖV99] M.T.Özsu and P.Valdurize, "*Principles of distributed database systems*". Second 2nd ed.,Upper Saddle River. NJ: Prentice-Hall 1999.

[Ozs80] Z. Ozsoyoglu, "*Query processing in distributed relational database*", Ph.D. dissertation, University of Alberta, 1980.

[PC90] W.Perrizo and C. S. Chen, "*Composite semijoin in distributed query processing*", Information Sciences, Vol.50, 197-218, 1990.

[RK91] N.Roussopoulos and H.Kang, *"A pipelined n-way join algorithm based on the 2-way semijoin program"*, IEEE Transaction on Knowledge and Data Engineering, 3(4), 486-495, 1991.

[Seg91] A.Segev, *"A global heuristic for distributed query optimization"*. Information Sciences, Vol.54, 67-88, 1991.

[TC92] J.C.R Tseng and A.L.P. Chen, *"Improving distributed query processing by hash-semijoins"*, Journal of Information Science and Engineering, Vol.8, 525-540, 1992.

[WLC93] C. Wang, V.O.K. Li, and A.L.P Chen, "*One-shot semijoin execution strategies for processing distributed queries*", Computer Systems Science Engineering, Vol.4, 245-253, 1993.

[WCS92] C. Wang, A.L.P Chen, and S.-C.Shyu, *"A parallel execution method for minimizing distributed query response time"*, IEEE Transactions on Parallel and Distributed Systems, 3(3):325-332, 1992.

[YL89] H.Yoo and S. Lafortune, *"An intelligent search method for query optimization by semijoins"*, IEEE Transactions on Knowledge and Data Engineering, 1(2): 22-237, 1989.

[WC96] C. Wang, M-S. Chen, "*On the Complexity of Distributed Query Optimization*", IEEE Trans. on Knowledge and Data Engineering, 8(4), 650-662, 1996.

VITA AUCTORIS

The author was born in Chang Chun, Ji Lin, People's Republic of China. She received her first professional education in Computer Science at Ji Lin University in 1981. She served as a director with the Engineer at Computing Center in Northeast Normal University in China until she came to Canada. She is currently a candidate for the Master's degree in Computer Science at the University of Windsor and hopes to graduate in Fall 2000.